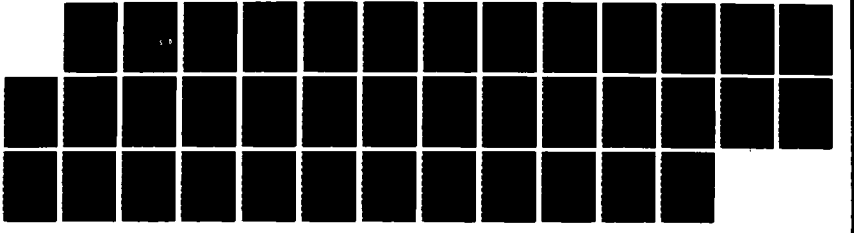
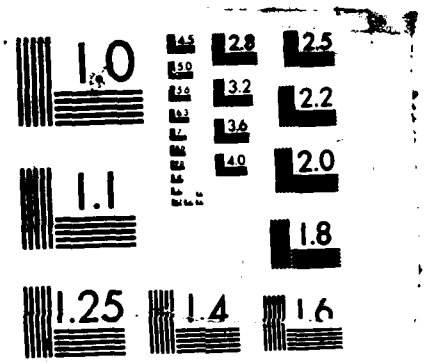
AD-A180 062 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT CAP INDUSTRY LTD CAPTA (U) NATIONAL COMPUTING CENTRE LTD MANCHESTER (ENGLAND) 11 DEC 86 1/1

UNCLASSIFIED F/G 12/5 NL





UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

1. REPORT NUMBER | 2. GOVT ACCESSION NO.	READ INSTRUCTIONS BEFORE COMPLETEING FORM 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: CAP Industry Ltd, CAPTACS-E286, V1.0 Host: DEC VAX 8800, Target: Intel 80286	5. TYPE OF REPORT & PERIOD COVERED 11 DEC 1986 to 11 DEC 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) The National Computing Centre Ltd	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS The National Computing Centre Ltd Oxford Road, Manchester M1 7ED, United Kingdom	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 11 DEC 1986 13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* The National Computing Centre Ltd	15. SECURITY CLASS *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Ada* Compiler Validation Summary Report:

Compiler Name: CAPTACS-E286, V1.0

Host:
DEC VAX 8800 under
VMS
4.4

Target:
INTEL 80286 under
no operating system.

Testing Completed 11th December 1986
Using ACVC 1.8

This report has been reviewed and is approved.

The National Computing Centre Ltd
Vony Gwillim
Oxford Road
Manchester
M1 7ED

for JFK

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

AVF Control Number: AVF-VSR-90502/09

Ada* COMPILER
VALIDATION SUMMARY REPORT:
CAP Industry Ltd
CAPTACS-E286, V1.0
Host: DEC VAX 8800
Target: INTEL 80286

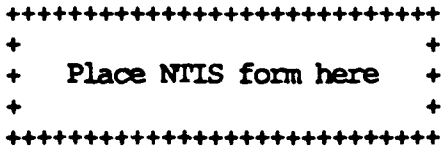
Completion of On-Site Testing:
11th December 1986

Prepared By:
The National Computing Centre Ltd
Oxford Road
Manchester
M1 7ED
United Kingdom

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.
USA

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).


Place NTIS form here

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the CAPTACS-E286, V1.0 using Version 1.8 of the Ada* Compiler Validation Capability (ACVC). The CAPTACS-E286 hosted on a DEC VAX 8800 operating under VMS, V4.4. Programs processed by this compiler may be executed on an INTEL 80286.

On-site testing was performed 8th December 1986 through 11th December 1986 at CAP Industry Ltd., Reading, under the direction of The National Computing Centre (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation were not processed. After the 2210 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 88 of the processed tests determined to be inapplicable; The remaining 2122 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	252	334	243	161	97	136	262	109	32	217	177	2122
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	73	86	4	0	0	3	0	21	0	1	56	258
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard.
- To determine that the implementation-dependent behaviour is allowed by the Ada Standard

Testing of this compiler was conducted by NCC under the direction of the AVF according to policies and procedures established by the Ada Validation Organisation (AVO). On-site testing was conducted from 8th December 1986 through 11th December 1986 at CAP Industry Ltd., Reading.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. # 552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organisations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
The National Computing Centre Ltd
Oxford Road
Manchester
M1 7ED
United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementer's Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The National Computing Centre Ltd. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — A test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn — A test found to be incorrect and not used to check conformity to test the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capabilities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimization allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: CAPTACS-E286, V1.0

ACVC Version: 1.8

Certification Expiration Date: 12 December 1987

Host Computer:

Machine :	DEC VAX 8800
Operating System:	VMS 4.4
Memory Size:	32M bytes

Target Computer:

Machine :	INTEL 80286 chip with - one 80287 numeric processor chip for floating point operations - one 8254 interval timer chip - two 8259A interrupt controller chips tested on an iSBC286/12 board
Operating System:	no operating system.
Memory Size:	1M bytes RAM

Communications Network: RS232C Connector

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types LONG_INTEGER, and LONG_FLOAT, in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array Types.

  An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/ or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises STORAGE_ERROR when the array objects are declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

  During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

  In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

  In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

  All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Functions

  An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in

the same immediate scope, or it may reject the function declaration. If it accepts the function declarations, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declarations. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE and 'STORAGE_SIZE for tasks and 'SMALL clauses; it rejects 'STORAGE_SIZE for collections. Enumeration representation clauses appear not to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests CA3004E and CA3004F.)

. Input/Output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

This implementation implements the input/output package TEXT_IO and SEQUENTIAL_IO for specific named sequential devices only, and package DIRECT_IO as an effective "null" package. In the absence of NAME_ERROR (file name > 12 characters long), any attempt to create or open a named or temporary file (other than the specific named devices) raises USE_ERROR. Other exceptions are raised, as appropriate, as specified in Chapter 14 of the Language Reference Manual.

. Generics.

Generic subprogram declarations and bodies cannot be compiled in separate compilations. (See test CA2009F.)

Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of CAPTACS-E286 was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 258 tests were inapplicable to this implementation, and that the 2122 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	67	862	1134	17	10	34	2122
Failed	0	0	0	0	0	0	0
Inapplicable	2	5	235	0	3	13	258
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	102	252	334	243	161	97	136	262	109	32	217	177	2122
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	73	86	4	0	0	3	0	21	0	1	56	258
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 258 tests were inapplicable for the reasons indicated:

- C324001D, B52004E, B55B09D, and C55B07B use SHORT_INTEGER which is not supported by this compiler.

. C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.

. C55B16A makes use of an enumeration representation clause containing noncontiguous values which is not supported by this compiler.

. B86001DT requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

. C87B62B uses a length clause to specify the collection size for an access type which is not supported by this compiler. The length clause is rejected during compilation.

. CA1012A, BA1011C, CA2009C, BC3205D and CA2009F, compile generic parts in separate compilation files. Separate compilation of generic specifications and bodies is not supported by this compiler.

. LA5008A..H, J, M and N (11 tests) compile generic parts in separate compilation files. Separate compilation of generic specifications and bodies is not supported by this compiler.

. CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

. CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

. AE2101C, CE2201D, and CE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types which is not supported by this compiler.

. AE2101H, CE2401D use instantiation of package DIRECT_IO with unconstrained array types which is not supported by this compiler.

. CE2210A uses dynamic creation and resetting of files which is not supported by this compiler.

. CE3110A uses dynamic creation and deletion of files which is not supported by this compiler.

- This implementation raises USE_ERROR when an attempt is made to create or open a named file using TEXT_IO or SEQUENTIAL_IO, or any file using DIRECT_IO. As a result, the following tests are not applicable:

CE2102D	CE2111A	CE2408A
CE2102E	CE2111B	CE2409A
CE2102F	CE2111C	CE2410A
CE2102G	CE2111D	CE3102B
CE2102I	CE2111E	CE3107A
CE2102J	CE2111G	CE3108A
CE2104A	CE2111H	CE3108B
CE2104B	CE2201C	CE3112B
CE2104C	CE2401A	CE3114A
CE2104D	CE2401B	CE3114B
CE2107A	CE2401C	CE3115A
CE2107B	CE2401E	CE3305A
CE2107F	CE2401F	CE3704F
CE2110A	CE2404A	CE3704M
CE2110B	CE2405B	CE3704N
CE2110C	CE2406A	CE3905L

EE3102C

- The following 170 tests make use of floating-point precision that exceeds the maximum of 15 supported by the implementation:

C24113L..Y (14 tests)
C35705L..Y (14 tests)
C35706L..Y (14 tests)
C35707L..Y (14 tests)
C35708L..Y (14 tests)
C35802L..Y (14 tests)
C45241L..Y (14 tests)
C45321L..Y (14 tests)
C45421L..Y (14 tests)
C45424L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subsets that can be processed.

Splits were required for 28 Class B tests.

B55A01A	B95077A	BA3007B
B56001H	B97101A	BA3008A
B71001B..F	B97101E	BA3008B
B71001I	BA1101C	BA3013A
B71001K	BA3006A	BC10AEB
B71001N..R	BA3006B	BC1202E
B71001U		
B71001W		

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by CAPTACS-E286, was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of CAPTACS-E286 using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a DEC VAX 8800 host operating under VMS, V4.4, and a INTEL 80286 target with no operating system. The host and target computers were linked via RS232C connector.

A magnetic tape containing all tests was taken on site by the validation team for processing. Tests that make use of values that are specific to an implementation were customised on-site after the magnetic tape was loaded. The validation package REPORT was modified to use package REPORT_IO instead of TEXT_IO. The package REPORT_IO is a subset of TEXT_IO that directs standard input and output to the host computer via RS232C connector. The results were stored in a file on the host computer and then printed. Tests requiring splits during the prevalidation testing were not included in their split form on the magnetic tape. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the DEC VAX 8800, and all executable tests were run on the INTEL 80286. Object files were linked on the host computer, and executable images were transferred to the target computer via RS232C connector. Results were printed from the host computer.

The compiler was tested using command scripts provided by CAP Industry Ltd., and reviewed by the validation team. The following options were in effect for testing:

Option	Effect
/LIST	Specifies a source listing is to be produced.

Tests were compiled, linked and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at AVF. The listings examined on-site by the validation team were also archived.

3.7.3 TEST SITE

The validation team arrived at CAP Industry Ltd., Reading, on 8th December 1986 and departed after testing was completed on 11th December 1986.

APPENDIX A

COMPLIANCE STATEMENT

CAP Industry Ltd has submitted the following compliance statement concerning the CAPTACS-E286.

Compliance Statement

Base Configuration:

Compiler: CAPTACS-E286, V1.0

Test Suite: Ada* Compiler Validation Capability, Version 1.8

Host Computer:

Machine: DEC VAX 8800

Operating System: VMS
4.4

Target Computer:

Machine: INTEL 80286 chip with

- one 80287 numeric processor chip for floating point operations
- one 8254 interval timer chip
- two 8259A interrupt controller chips

tested on an iSBC286/12 board.

Operating System: no operating system

Communications Network: RS232C connector

CAP Industry Ltd has made no deliberate extensions to the Ada language standard.

CAP Industry Ltd agrees to the public disclosure of this report.

CAP Industry Ltd agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Godfrey Draper Date: 17|12|86

CAP Industry Ltd
Godfrey Draper
Ada Product Development Manager

*Ada is registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the CAPTACS-E286, V1.0 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
Package STANDARD is

   ...

   type INTEGER is -32768 .. 32767                          ;
   type LONG_INTEGER is range -2147483648 .. 2147483647  ;

   type FLOAT is digits 6  range +/-16#0.FFFFFF#E32       ;
   type LONG_FLOAT is digits 15 range +/-16#0.FFFFFFFFFFFFF8#E256;

   type DURATION is delta 2**-14 range -86400.0 ..86400.0 ;
    --  DURATION'SMALL =  2**-14                                .


end STANDARD;
```

F APPENDIX F

The Ada language definition allows for certain target dependencies in a controlled manner. This appendix, called Appendix F as prescribed in the LRM, describes implementation-dependent characteristics of the CAPTACS-E286 system.

F.1 Implementation-Dependent Pragmas

There is one implementation-dependent pragma called COMMENT. The pragma is used for embedding a sequence of characters into the object code. The syntax is:

```
pragma COMMENT ( <string-literal> );
```

where: <string-literal> represents the characters to be embedded in the object code.

Pragma COMMENT may appear at any location within the source code of a compilation unit. Any number of comments may be entered into the object code using this method.

F.2 Implementation-Dependent Attributes

There are no implementation-dependent attributes.

F.3 Package SYSTEM

The current specification of the package is provided below.

```
package SYSTEM is

    type SEG_OFFSET is new INTEGER;

    type SEG_SELECTOR is new INTEGER;

    type ADDRESS is private;

    type SUBPROGRAM_VALUE is private;

    type NAME is (CAPTACS_E286);

    SYSTEM_NAME : constant NAME := CAPTACS_E286;

    STORAGE_UNIT : constant := 8;
    MEMORY_SIZE : constant := 2**24;
```

```
    --System-Dependent Named Numbers:

    MIN_INT       : constant := -(2**31);
    MAX_INT       : constant := (2**31) - 1;
    MAX_DIGITS    : constant := 15;
    MAX_MANTISSA  : constant := 31;
    FINE_DELTA    : constant := 1.0 / (2**( MAX_MANTISSA - 1 ));
    TICK          : constant := 1.0 / (2 ** 10);

    --Other System-Dependent Declarations:

    subtype PRIORITY is INTEGER range 0..15;

private

-- Types ADDRESS and SUBPROGRAM_VALUE are system-dependent

end SYSTEM;
```

F.4 Representation Clauses

The CAPTACS-E286 Compiler supports the following representation clauses:

- Length Clauses :

    1) for the attribute 'storage_size for task types.

    2) for the attribute 'size. The value specified is checked to be sufficient, but otherwise ignored.

    3) for the attribute 'small.

F.5 Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components.

F.6 Address Clause Expression Interpretation

Expressions that appear in Address specifications are interpreted as the first storage unit of the object.

F.7 Unchecked Conversion Restrictions

Unchecked conversions are allowed between types (or subtypes) T1 and T2 provided that:

1) they have the same static size.

2) they are not unconstrained array types.

3) they are not private (unless they are subtypes of or are derived from type SYSTEM.ADDRESS).

F.8 Implementation-Dependent Characteristics of the I/O Packages

1. Instantiations of DIRECT_IO and SEQUENTIAL_IO are supported with the following exceptions:

   * unconstrained array types

   * unconstrained types with discriminants without defaults

2. There is no support for a file system. DIRECT_IO always raises a run-time exception. SEQUENTIAL_IO and TEXT_IO are supported only for sequential devices.

   Any attempt to create or open a named or temporary file (other than the specific named sequential devices) raises NAME_ERROR if the file name is greater than 12 characters long, and USE_ERROR otherwise.

3. In DIRECT_IO, the type COUNT is defined as follows:

```
type COUNT is range 0..2147483647;
```

4. In TEXT_IO, the type COUNT is defined as follows:

```
type COUNT is range 0..2147483645;
```

5. In TEXT_IO, the subtype FIELD is defined as follows:

```
subtype FIELD is INTEGER range 0..1000;
```

6. Package LOW_LEVEL_IO is defined for the following types:

```
device_type:

    type PORT_ADDRESS is new INTEGER;

data_types:
```

```
type DATA_BYTE is range 0..255;
type DATA_WORD is range INTEGER'first..INTEGER'last
type DATA_BYTE_ARRAY is array (NATURAL range <>) of DATA_BYTE;
type DATA_WORD_ARRAY is array (NATURAL range <>) of DATA_WORD;
```

F.9 Compilation of Generic Units

The declaration and body of a generic unit must be submitted as a single compilation (ie must be in the same source file).

F.10 Package MACHINE_CODE

Package MACHINE_CODE is not supported.

F.11 Language-Defined Pragmas

The language-defined pragmas ELABORATE, INTERFACE (ASSEMBLY_INTERFACE), PRIORITY and SUPPRESS are fully supported. The other language-defined pragmas, if included in Ada source, will have no effect.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

NAME AND MEANING	VALUE
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	A....A1 |----| 199 characters
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	A....A2 |----| 199 characters
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	A....A3A....A |----| |----| 99 100 characters
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	A....A4A....A |----| |----| 99 100 characters
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that is is the size of the maximum line length.	0....0298 |----| 197 characters
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value of 690.0, and has enough leading zeroes to be the size of the maximum line length.	0....069.0E1 |----| 194 characters

NAME AND MEANING	VALUE
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	180 blanks
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483645
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz !$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST	1000
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long if no invalid characters exist.	X}]!@#=$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long if no wild card characters exists.	XYZ*
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	BAD-CHARACTER *^/%

NAME AND MEANING	VALUE
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	A....A |----| 120 characters
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	32767
LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	200
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	No such type use LONG_INTEGER

NAME AND MEANING	VALUE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non_ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

. C32114A: An unterminated string literal occurs at line 62.

. B33203C: The reserved word "IS" is misspelled at line 45.

. C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.

. C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

. B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

. C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.

. B45116A: ARRPRIBL 1 and ARRPRIBL 2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

. C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

. B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.

. B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

. B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

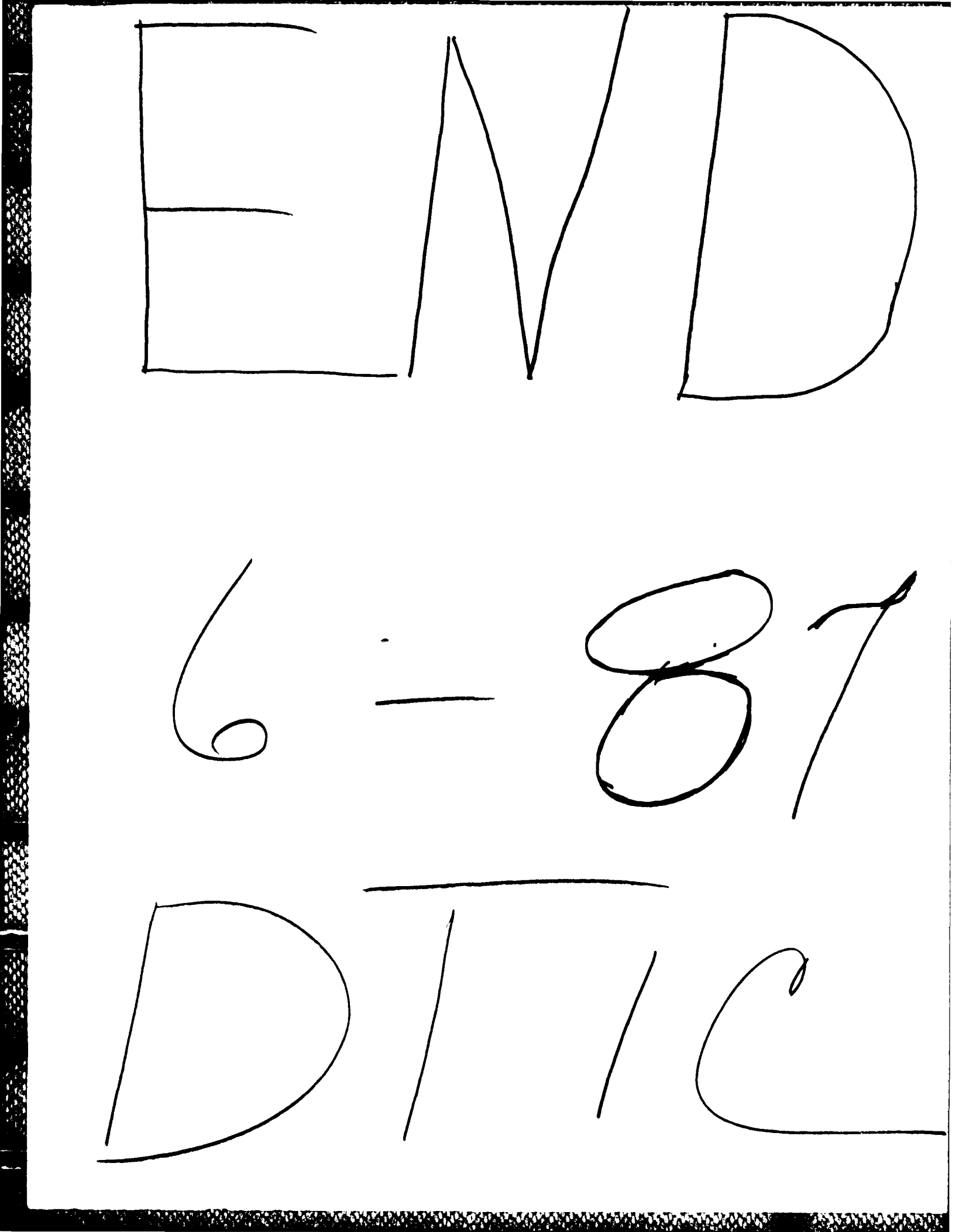
. C87B50A: The call of "/=" at line 31 requires a use clause for package A.

. C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

. C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

. CA3005A..D: (4 tests) No valid elaboration order exists for these tests.

. BC3204C: The body of BC3204C0 is missing.


END
6-87
DTIC